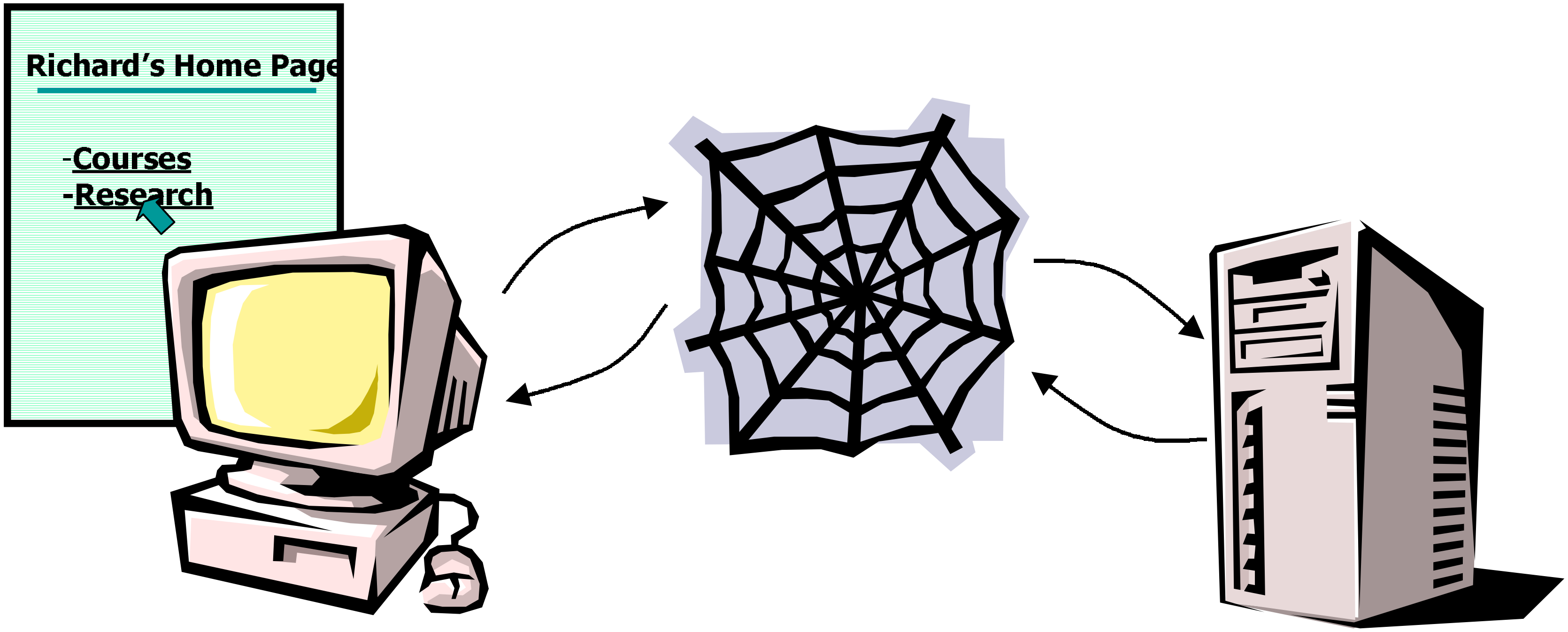
Communication

Sockets (Haviland – Ch. 10)

Simple Web Request

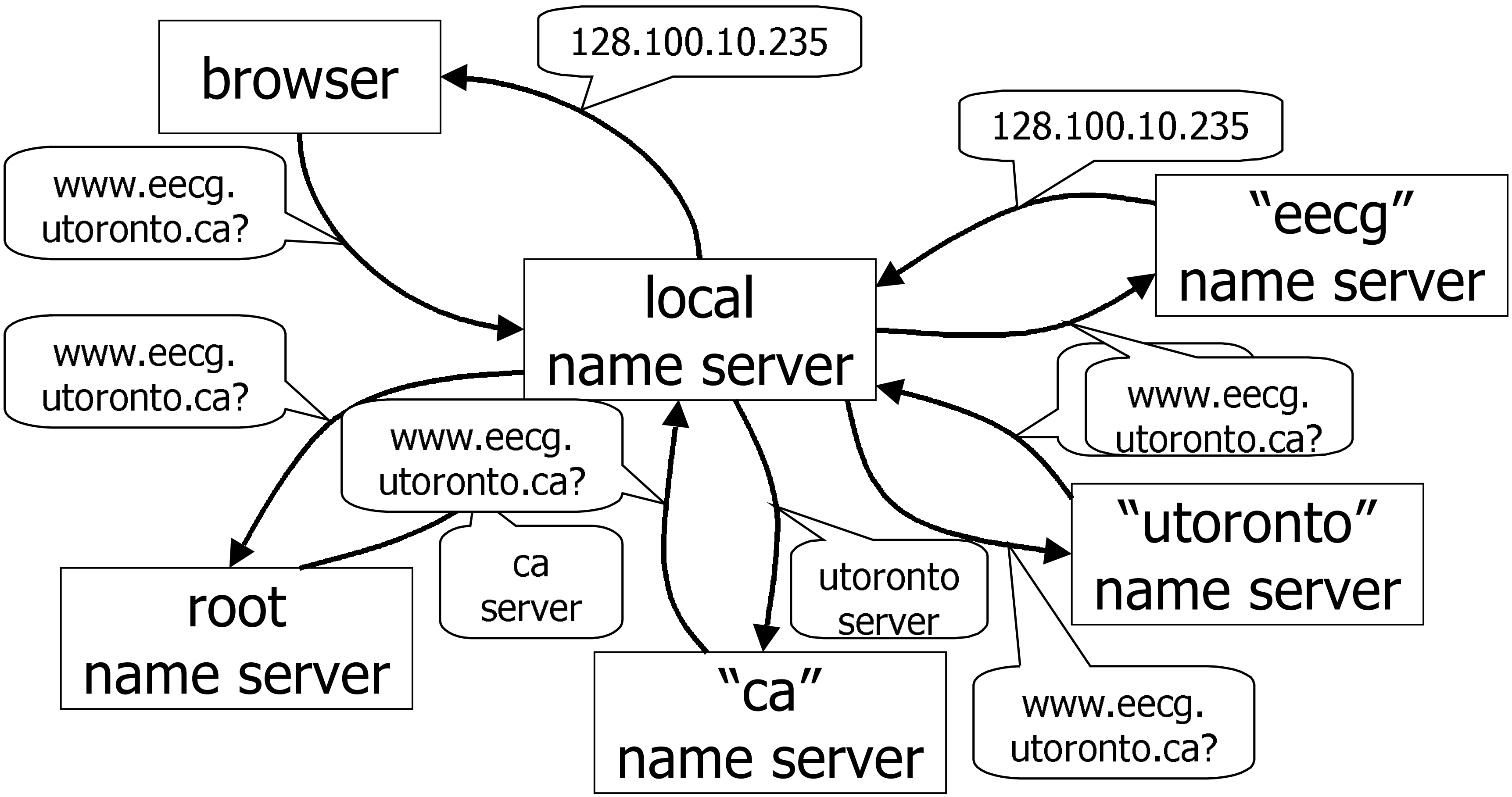


How do we find the server?

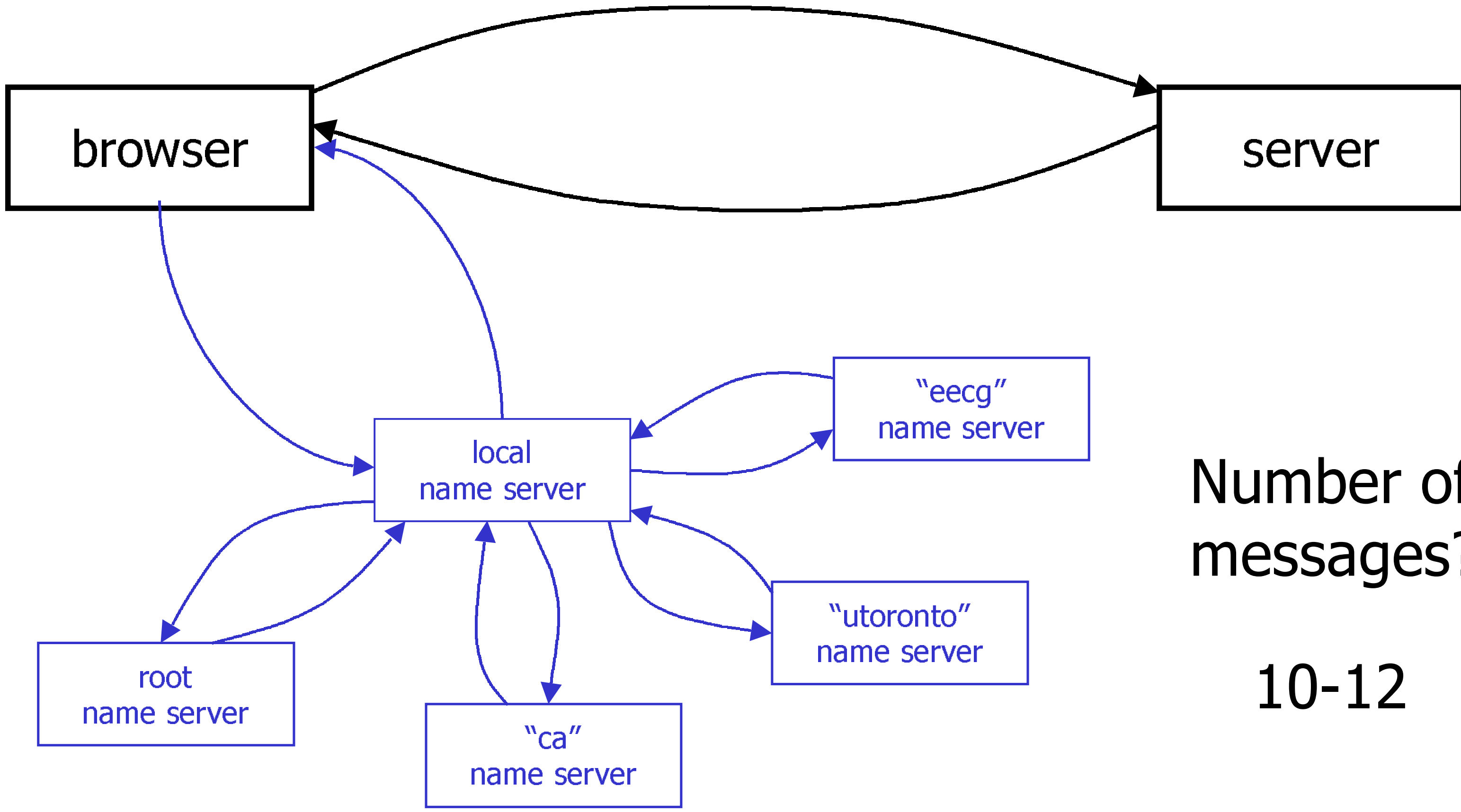
- Every computer on the Internet has an Internet address.
- Called an IP address (Internet Protocol)
- An IP address is four 8-bit numbers separated by dots.

www.eecg.toronto.edu = 128.100.10.235

Domain Name Servers



This is getting complicated!



Number of messages?

10-12

Protocols

Invoice:
Customer: John Doe
Order No: 5379

Qty:		Unit Price	Total
1	Athalon	219.00	219.00
2	128 MB	149.95	299.90
		Subtotal	518.90
		Tax	77.84
		TOTAL	596.74

John Doe Feb 18, 2004

Payable to: CPUS are us $596.74
Five hundred ninety six 74/100

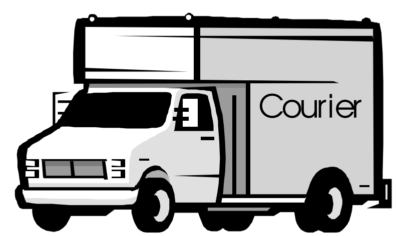
____

CPUS are us

John Doe
Dept. of Computer Science
University of Toronto

John Doe

CPUS are us
0 College Street
Toronto Ontario M5S 3G4

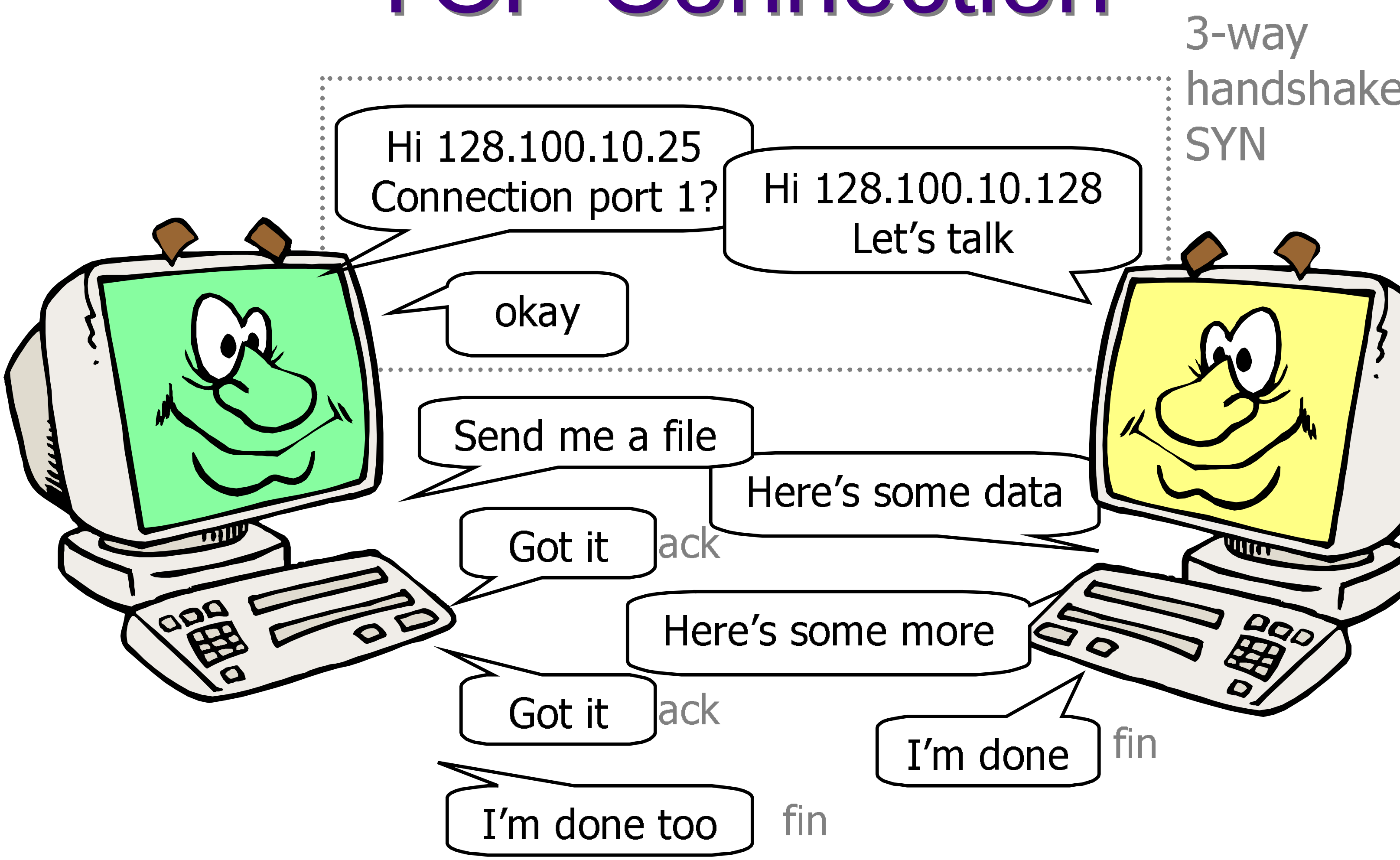


TCP/IP

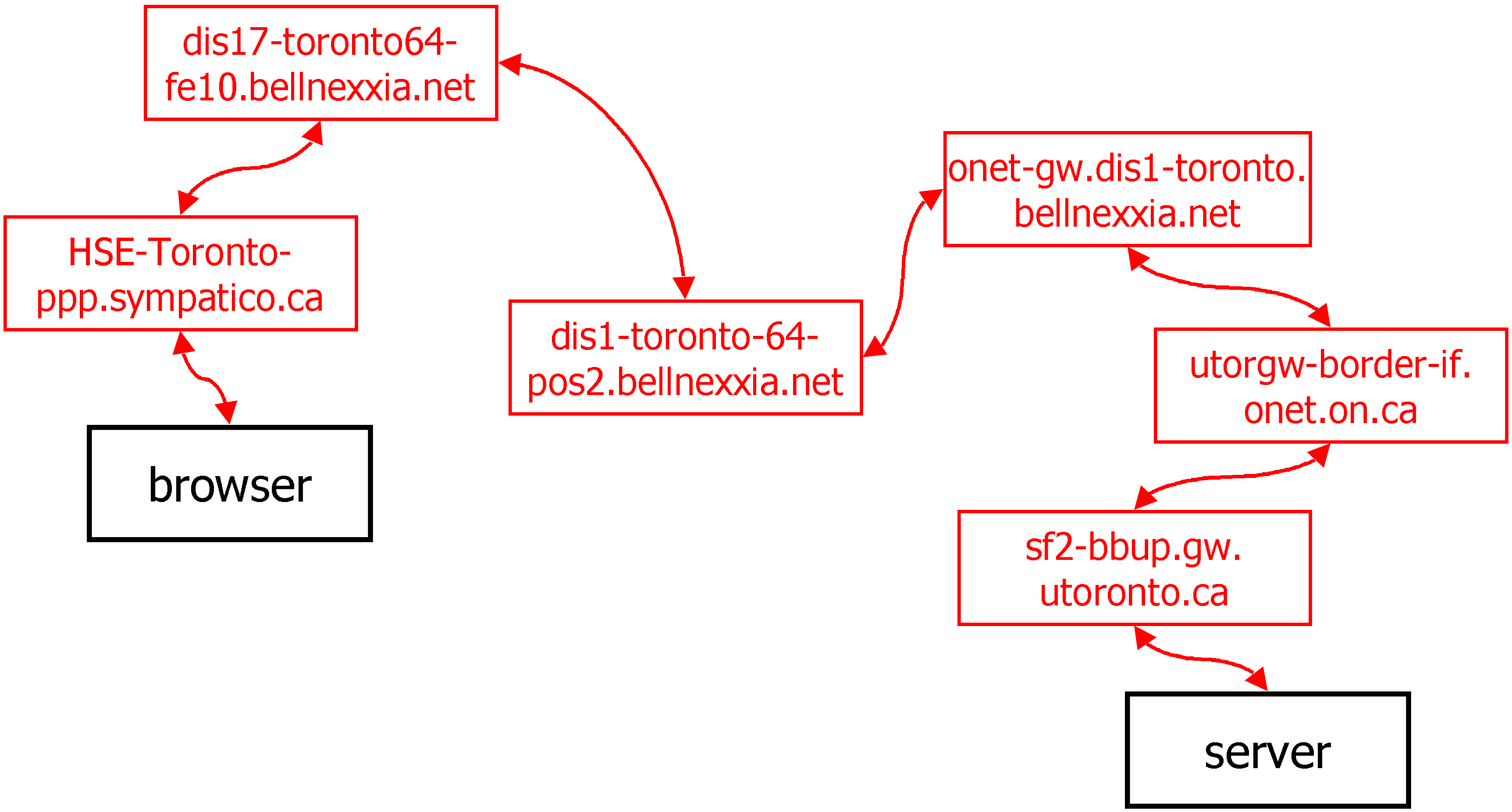
- Transmission Control Protocol.
- Tells us how to package up the data.

<table>
<tr><td colspan="2">source address</td><td>dest. address</td></tr>
<tr><td>bytes</td><td>ack</td><td>port</td></tr>
<tr><td colspan="3">data</td></tr>
</table>

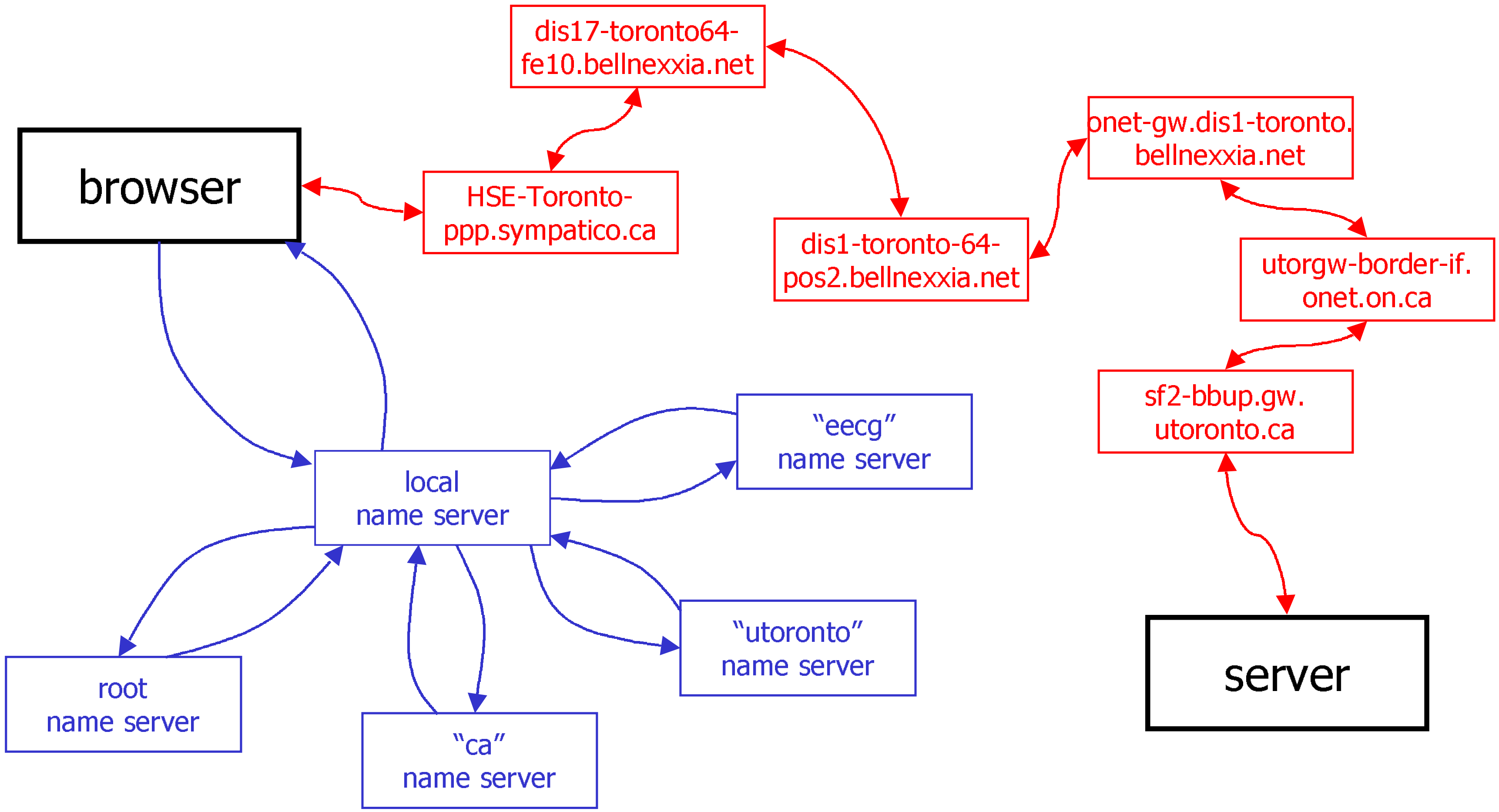
TCP Connection



Routing


dis17-toronto64-
fe10.bellnexxia.net
HSE-Toronto-
ppp.sympatico.ca
browser
dis1-toronto-64-
pos2.bellnexxia.net
onet-gw.dis1-toronto.
bellnexxia.net
utorgw-border-if.
onet.on.ca
sf2-bbup.gw.
utoronto.ca
server

Putting it together

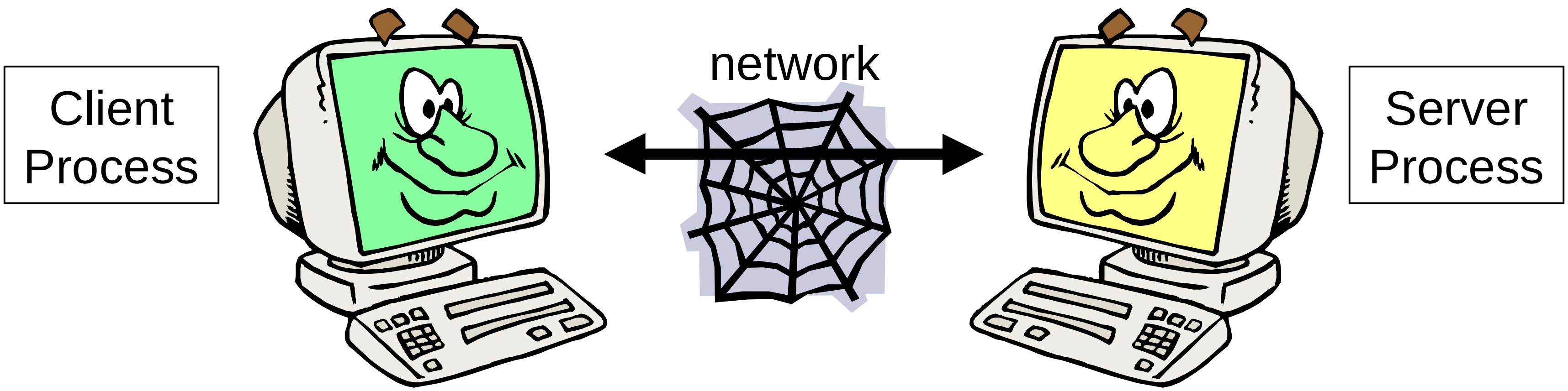


How many messages?

- It depends on the size of the web page we retrieve.
- If the web page is 75 Kbytes (small!) it will be broken up into 103 IP packets.
- Remember DNS took 10 messages

10 + 103 x 7 hops = 731 messages!

The Big Picture

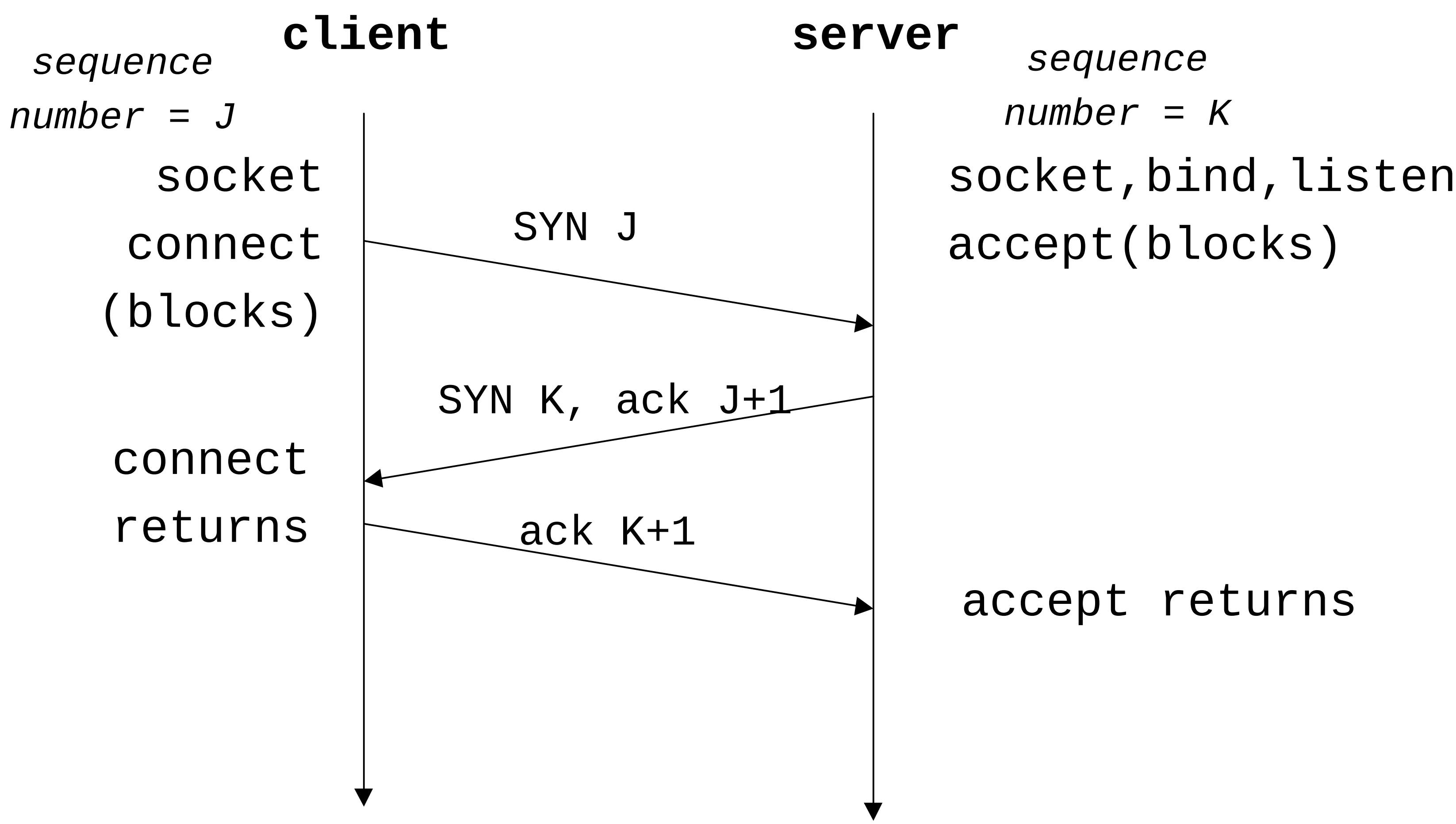


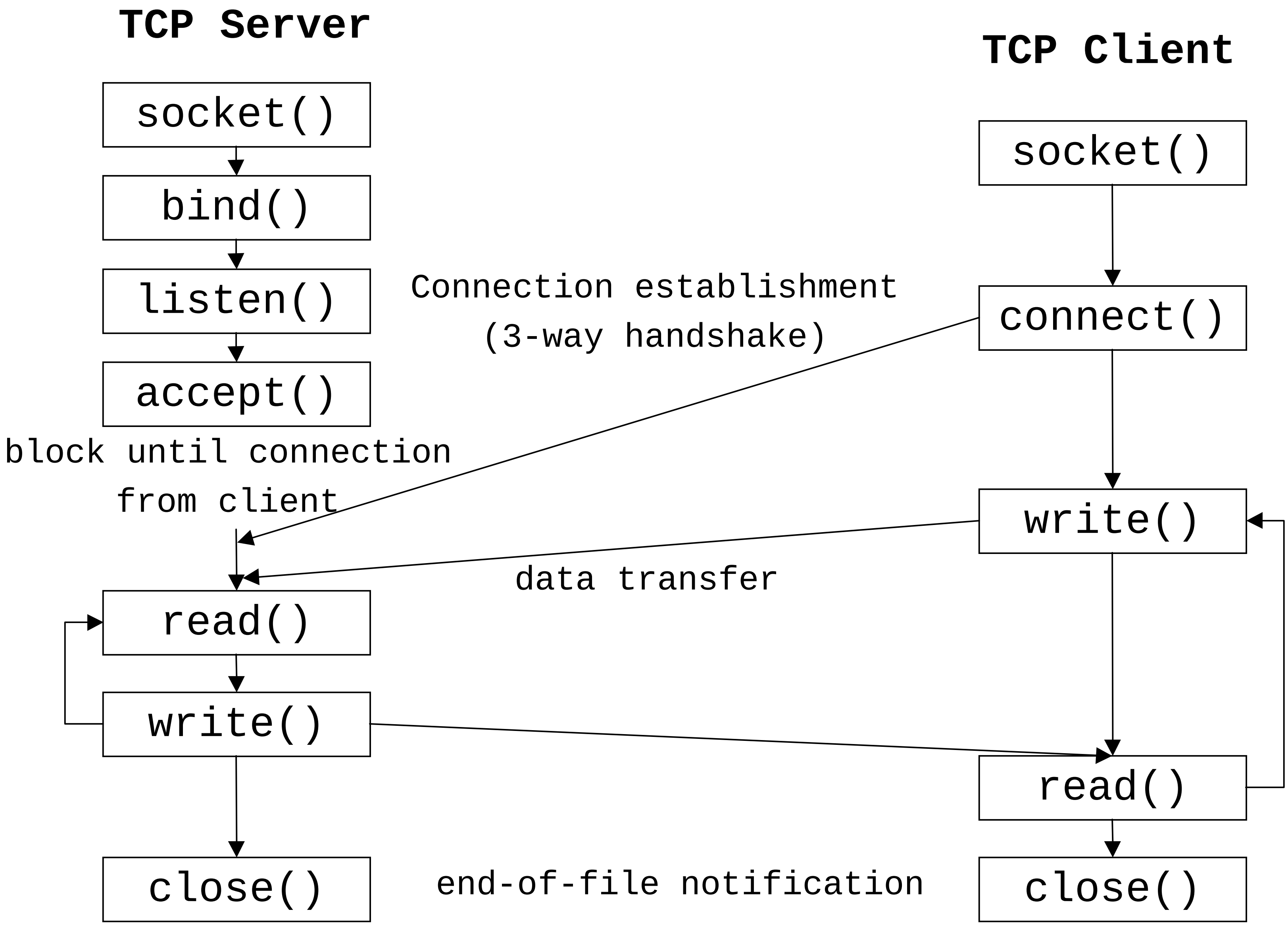
- **Client-Server model**: a client process wants to talk to a server process
- Client must find server - **DNS lookup**
- Client must find process on server - **ports**
- Finally **establish a connection** so two processes can talk

Sockets

- One form of communication between processes.
- Similar to pipes, except sockets can be used between processes on different machines.
- Use file descriptors to refer to sockets.
- Built on top of TCP layer

TCP: Three-way handshake




TCP Server
TCP Client
socket()
bind()
listen()
accept()
block until connection
from client
Connection establishment
(3-way handshake)
socket()
connect()
write()
data transfer
read()
write()
read()
close()
end-of-file notification
close()

Connection-Oriented

Server

- Create a socket: `socket()`
- Assign a name to a socket: `bind()`
- Establish a queue for connections: `listen()`
- Get a connection from the queue: `accept()`

Client

- Create a socket: `socket()`
- Initiate a connection: `connect()`

Socket Types

- Two main categories of sockets
  - UNIX domain: both processes on the same machine
  - INET domain: processes on different machines
- Three main types of sockets:
  - `SOCK_STREAM`: the one we will use
  - `SOCK_DGRAM`: for connectionless sockets
  - `SOCK_RAW`

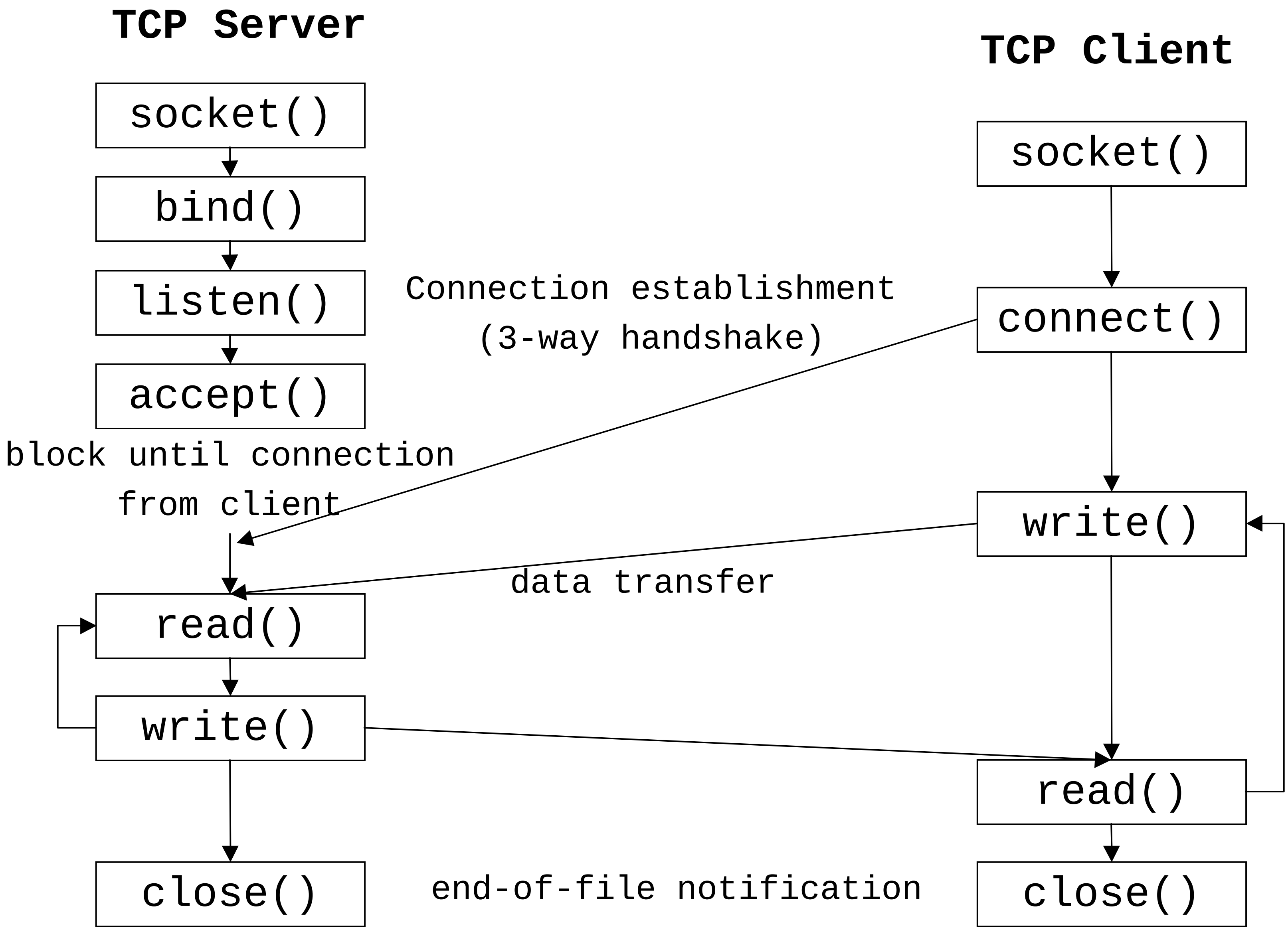
Addresses and Ports

- A socket pair is the two endpoints of the connection.
- An endpoint is identified by an IP address and a port.
- IPv4 addresses are 4 8-bit numbers:
  - 128.100.31.200 = werewolf
  - 128.100.31.201 = seawolf
  - 128.100.31.202 = skywolf
- Ports
  - because multiple processes can communicate with a single machine we need another identifier.

More on Ports

www.iana.org

- Well-known ports: 0-1023
  - 80 = web
  - 22 = ssh
  - 23 = telnet
  - 21 = ftp
  - 25 = smtp (mail)
  - 194 = irc
- Registered ports: 1024-49151
  - 2709 = supermon
  - 26000 = quake
- Dynamic (private) ports: 49152-65535
  - You should pick ports in this range to avoid overlap


TCP Server
TCP Client
socket()
bind()
listen()
accept()
block until connection
from client
read()
write()
close()
socket()
connect()
write()
read()
close()
Connection establishment
(3-way handshake)
data transfer
end-of-file notification

Server side

```
int socket(int family, int type,
           int protocol);
```

- family specifies protocol family:
  - `PF_INET` – IPv4
  - `PF_LOCAL` – Unix domain
- type
  - `SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_RAW`
- protocol
  - set to 0 except for RAW sockets
- returns a socket descriptor

bind to a name

```
int bind(int sockfd,
      const struct sockaddr *servaddr,
      socklen_t addrlen);
```

- sockfd – returned by socket
- `struct sockaddr_in {`

```
    short           sin_family; /*AF_INET */
    u_short         sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8]; /*filling*/
  };
```

- `sin_addr` can be set to `INADDR_ANY` to communicate with any host

Set up queue in kernel

```
int listen(int sockfd, int backlog)
```

- after calling `listen`, a socket is ready to accept connections
- prepares a queue in the kernel where partially completed connections wait to be accepted.
- `backlog` is the maximum number of partially completed connections that the kernel should queue.

Complete the connection

```
int accept(int sockfd,
        struct sockaddr *cliaddr,
        socklen_t *addrlen);
```

- blocks waiting for a connection (from the queue)
- returns a new descriptor which refers to the TCP connection with the client
- `sockfd` is the listening socket
- `cliaddr` is the address of the client
- reads and writes on the connection will use the socket returned by accept

Client side

- `socket()` – same as server, to say “how” we are going to talk

```
int connect(int sockfd,
      const struct sockaddr *servaddr,
      socklen_t addrlen);
```

- the kernel will choose a dynamic port and source IP address.
- returns 0 on success and -1 on failure setting errno.
- initiates the three-way handshake.

inetclient.c

> most error checking is omitted in this example

```
int soc;
struct hostent *hp;
struct sockaddr_in peer;

peer.sin_family = AF_INET;
peer.sin_port = htons(PORT);
/* fill in peer address */
hp = gethostbyname(argv[1]);
peer.sin_addr = *((struct in_addr *)hp->h_addr);
/* create socket */
soc = socket(PF_INET, SOCK_STREAM, 0);
/* request connection to server */
if (connect(soc, (struct sockaddr *)&peer, sizeof(peer))
       == -1) {
  perror("client:connect"); close(soc); exit(1);
}
write(soc, "Hello Internet\n", 16);
read(soc, buf, sizeof(buf));
printf("SERVER SAID: %s\n", buf);
close(soc);
```

inetserver.c

full code for this example is on the webpage

```
struct sockaddr_in peer;
struct sockaddr_in self;
int soc, ns, k;
int peer_len = sizeof(peer);

self.sin_family = AF_INET;
self.sin_port = htons(PORT);
self.sin_addr.s_addr = INADDR_ANY;
bzero(&(self.sin_zero), 8);

peer.sin_family = AF_INET;
/* set up listening socket soc */
soc = socket(PF_INET, SOCK_STREAM, 0);
if (soc < 0) {
  perror("server:socket"); exit(1);
}

if (bind(soc, (struct sockaddr *)&self, sizeof(self))== -1){
  perror("server:bind"); close(soc); exit(1);
}
listen(soc, 1);
...
```

inetserver.c (concluded)

```
/* ... repeated from previous slide ...
  soc = socket(PF_INET, SOCK_STREAM, 0);
  bind(soc, (struct sockaddr *)&self, sizeof(self))== -1){
    perror("server:bind"); close(soc); exit(1);
  }
  listen(soc, 1);
... and now continuing ... */

  /* accept connection request */
  ns = accept(soc, (struct sockaddr *)&peer, &peer_len);
  if (ns < 0) {
    perror("server:accept"); close(soc); exit(1);
  }
  /* data transfer on connected socket ns */
  k = read(ns, buf, sizeof(buf));
  printf("SERVER RECEIVED: %s\n", buf);
  write(ns, buf, k);

  close(ns);
  close(soc);
```

Byte order

- Big-endian

91,329 =

A	A+1	A+2	A+3
00	01	64	C1

- Little-endian

91,329 =

A+3	A+2	A+1	A
00	01	64	C1

- Intel is little-endian, and Sparc is big-endian

Network byte order

- To communicate between machines with unknown or different “endian-ness” we convert numbers to network byte order (big-endian) before we send them.
- There are functions provided to do this:
  - `unsigned long htonl(unsigned long)`
  - `unsigned short htons(unsigned short)`
  - `unsigned long ntohl(unsigned long)`
  - `unsigned short ntohs(unsigned short)`

Sending and Receiving Data

- `read` and `write` calls work on sockets, but sometimes we want more control
- ```
  ssize_t send(int fd, const void *buf,
               size_t len, int flags);
  ```
  - works like `write` if `flags==0`
  - flags: `MSG_OOB`, `MSG_DONTROUTE`, `MSG_DONTWAIT`
- ```
  ssize_t recv(int fd, void *buf,
               size_t len, int flags);
  ```
  - flags: `MSG_OOB`, `MSG_WAITALL`, `MSG_PEEK`